Homework 4 Report Zhengxian Lin

Part I

I created a general-purpose simulator in simulator.py for this assignment. This simulator accepts three parameters as input, MDP, start state id, and terminal state id. The MDP is needed. For the other two, if there are not inputs for them, the default value of start state id is 0, and of the terminal state id is None, which does not means that it does not have terminal state. It means its terminal state is not just one state. Terminal state can be the state which has no state transitions and zero reward. It includes one function, take_action(a), which take an action as input, and return reward and next state.

Part II

Part a.

1. I created a simple MDP as test for this simulator in file "mdp_test.txt."

```
5 2

0.500 0.500 0.000 0.000 0.000
0.000 0.500 0.500 0.000 0.000
0.000 0.000 0.500 0.500 0.000
0.000 0.000 0.000 0.800 0.200
0.000 0.000 0.000 0.000 0.000

0.000 1.000 0.000 0.000 0.000
0.000 0.000 1.000 0.000 0.000
0.000 0.000 0.000 1.000 0.000
0.000 0.000 0.000 1.000 0.000
0.000 0.000 0.000 0.000 0.000

0 1
0 1
0 1
0 1
0 0
```

2. This simple MDP includes 5 states, 2 actions.
   For the S_0 to S_2, taking action a_0, the transition and reward are T(S_i, a_0, S_{i+1}) = 0.5, T(S_i, a_0, S_i) = 0.5, and R(S_i, a_0) = 0. For example, T(S_1, a_0, S_2) = 0.5, T(S_1, a_0, S_1) = 0.5, and R(S_1, a_0) = 0.
   For the S_0 to S_2, taking action a_1, the transition and reward are T(S_i, a_1, S_{i+1}) = 1.0, and R(S_i, a_1) = 1. For example, T(S_0, a_1, S_2) = 1.0, and R(S_0, a_1) = 1
   For the S_3, taking action a_0, the transition and reward are T(S_3, a_0, S_4) = 0.2, T(S_3, a_0, S_3) = 0.8, and R(S_3, a_0) = 0.

For the S_3, taking action a_1, the transition and reward are T(S_3, a_1, S_3) = 1.0, and R(S_3, a_1) = 1.
For the S_4, which is terminal state, there is no state transitions and zero reward.

3. This test MDP is quite easy to calculate its estimate of the value of a random policy from the start state:
   For the value of S_0, it depends on S_1, and the S_1 depends on S_2, it means the S_i depends on S_{i+1}, except the terminal state, since they have probability to go to next state using the random policy. For this simulation, the beta is 0.8
   There is bellman back equation:

$$V_s = R\big(S, \pi(S)\big) + \beta \sum_{s' \in S} T(S, \pi(S), S') * V_{s'}$$

   Thus, for S_0 to S_2, and the policy is random, so the probability of choosing a_0 or a_1 is same, 0.5. The value of those state is:

$$V_{s_i} = 0.5 * (0 + 0.8 * \big(0.5 * V_{s_i} + 0.5 * V_{s_{i+1}}\big) + 0.5 * (1 + 0.8 * \big(1 * V_{s_{i+1}}\big)$$

$$V_{s_i} = 0.75 * V_{s_i+1} + 0.625 \quad (\text{i} = 0 \sim 2)$$

   For the S_3, the value depends on the value of S_4. The S_4 is terminal state, so $V_{s_4} = 0$.

$$V_{s_3} = 0.5 * (0 + 0.8 * \big(0.8 * V_{s_3} + 0.2 * V_{s_4}\big) + 0.5 * (1 + 0.8 * \big(1 * V_{s_3}\big)$$

$$V_{s_3} = 0.5 + 0.72 * V_{s_3}$$

$$V_{s_3} \approx 1.79$$

   Once we get the value of S_3, we can get the value of S_2, S_1 and S_0. Finally, the estimate of the value of a random policy from the start state, which is S_0, is 2.20

4. Using the simulator for running test MDP to get the average reward of 100 episodes, the result is: 2.208246166949216. It also showed at the file, "results\resultOfPolicySimulation.txt"
5. For the optimal result, the action is a_1 for all of states, except the terminal state, because it will go to the S_3, and get the reward forever, and the reward of all state is 1, and transition is 1 too. So, the value of optimal is

$$V = 1 + 0.8 * (1 * V)$$

V = 5

Part b.

1. For the parking MDP, I change the parameters of two of them of assignment 3, why I did this will explain at next part. Description of the parking_mdp.py again here:
   The both MDP I design for n = 10, which means there are 10 spots of each line, A and B. In other word, there are 20 spots. Following the requirement on the pdf, the A[1] and B[1]

are handicap, and the possibility of occupied is decreased, and the reward of spots are increased except the A[1] and B[1], while the spot closer to the store.
There are three parameters of my MDP. The first one is n, which equals 10. The second one and third one is parameter set. They are same.
For the parameter set, paras = (p1, p2, p3, p4, p5, p6), the meaning of each sub parameter in the tuple is:
p1: The decreasing value of probability of occupation from the closer to the farther. For example, if p1 = 0.1, and the probability of occupation of A[2] is 0.9, then and the probability of occupation of A[3] is 0.8 = 0.9 – 0.1.
p2: The value of possibility of Occupation of Handicap for both A[1] and B[1]
p3: A tuple (p3-0, p3-1). p3-0 is a highest value of spots, which is A[2] and B[2]; the p3-1 is the decreasing value of reward of spots. While the spot is further, the reward is less. For example, the reward of A[2] = p3-0 = 100, and the p3-1 is 10. Then, the reward of A[3] is 100 – 10 = 90
p4: Driving cost. The cost of driving action
p5: Collision Cost. The cost of parking action at a spot which already had car p6: The cost or reward of parking action of Handicap spots, A[1] and B[1]

2. For these two MDP, para1 and para2, which means the parameters of MDP1 and MDP2, are same except the punishment of the collision. The value of first one is -100 and of the other one is -20000. For these two MDP, the first agent thinks that collision is ok for it, and I call it "Rude agent". The second one is quite reasonable, because we all know that the punishment of collision is very large. We always avoid collision. I call it "reasonable driver." I try a lot of different parking MDP for Part III, some drive cost is large, and reward of spot is small, like my assignment 3, and something handicap's punishment is low, but they are not enough to show the different ability of three reinforcement learning algorithm of Part III. Finally, I choose this two MDP, and I will show why they are the good example at Part III.
3. The result of this part. They also can be find at the file: "results\resultOfPolicySimulation.txt"

**********parking_mdp1[policy: parkingRandomPolicy] (p = 0.1)*********
Average reward: 329.15555166666684
****
**********parking_mdp1[policy: parkingAvoidingOccupuiedPolicy] (p = 0.1)********
Average reward: 138.25925533333313
****
**********parking_mdp1[policy: parkingSimpleImpovementPolicy] (p = 0.1)********
Average reward: 509.02592933333324
****

**********parking_mdp2[policy: parkingRandomPolicy] (p = 0.1)*********
Average reward: -6701.937040999995
****
**********parking_mdp2[policy: parkingAvoidingOccupuiedPolicy] (p = 0.1)********

Average reward: 154.84814066666658
****
**********parking_mdp2[policy: parkingSimpleImpovementPolicy] (p = 0.1)********
Average reward: 528.033335666667
****

For the first policy and second policy, they are described at the assignment pdf. For the third one, I improve it through forcing it to park if it arrive at the first four spot, except two handicap, which is A[2], A[3], A[4], B[2], B[3], B[4]. They will get a lot reward of them. The results of them are the average of 100 episodes.

Analysis:
For the parking MDP1, the interesting part is the average reward of avoiding occupied spot policy is less than the random policy. This is the correct result. Because the smallest reward of the spots is 100, and the punishment of collision is -100, so it is not afraid of avoiding collision. And once avoiding occupied spot, it may have to cost drive cost, which is -10. For the improvement policy, it is improved.

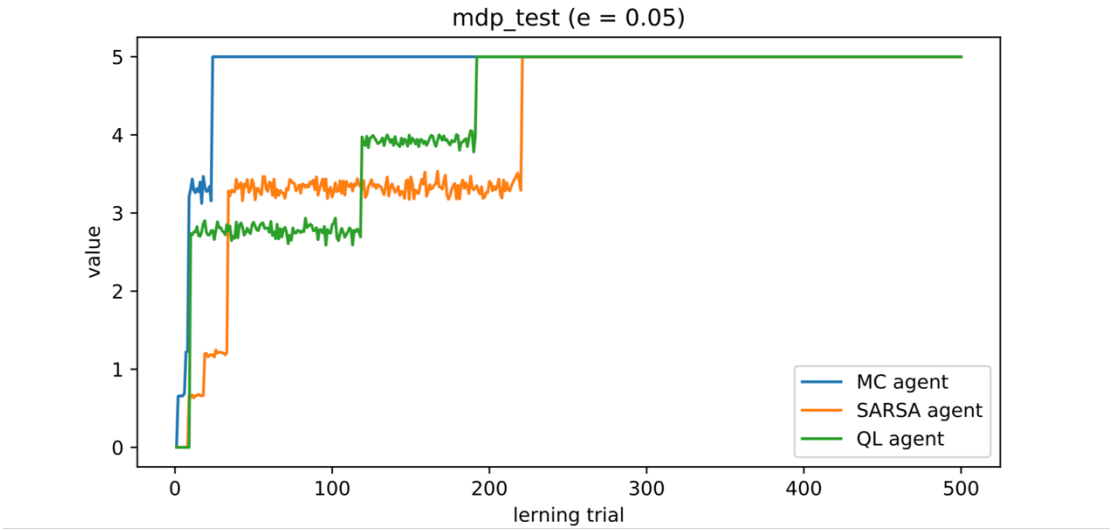
For the parking MDP2, the result is improved, which is we expected. Avoiding collision can get a lot improvement, since the punishment of collision is very large. The improvement policy definitely improves the reward.

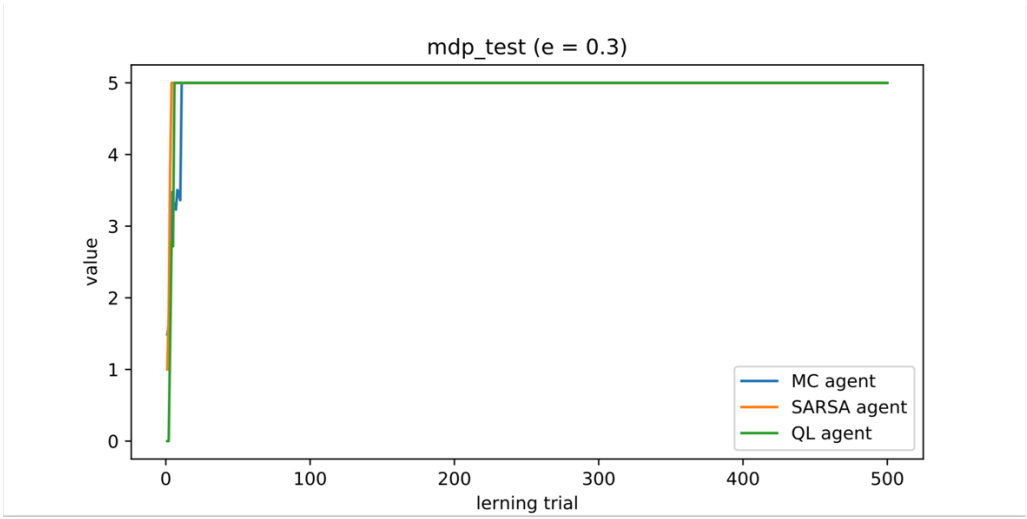
Part III

1. For the requirement of part III, I implemented those required three reinforcement learning agents, Monte Carol agent, SARSA agent, Q-learning agent. The beta is 0.999, and the learning rate α is 0.1. For parking MDP, the N is 50, which is the number of learning episodes. After learning 50 times, it runs 100 episodes to measures its performance, as same as part II. This process is one round, and each agent will do 500 rounds. So, there are 500 results, and they are showed at results folder, "resultOfMonte Carol Agent.txt", "resultOfSARSA Agent.txt", and "resultOfQ-learning Agent.txt." Also, they will be showed as figure at the below. For the simple MDP, the N is 1, which measure the performance after learning one episodes. Also, run it 500 rounds

2. There is one thing I mentioned at Part II, the reason why I select the parking MDP1 and MDP2. At first, when I run this part, I picked two MDP from my assignment 3. One's drive cost is low and spot reward is high. The other one's drive cost is high, and spot reward is high. However, the performance of Monte Carol agent and SARSA agent is bad for both of them. So, I try to find out why they are not good. Finally, I found out that this is because of the punishment is very high. So, I create two parking MDP, which's punishment is different, as research objects.

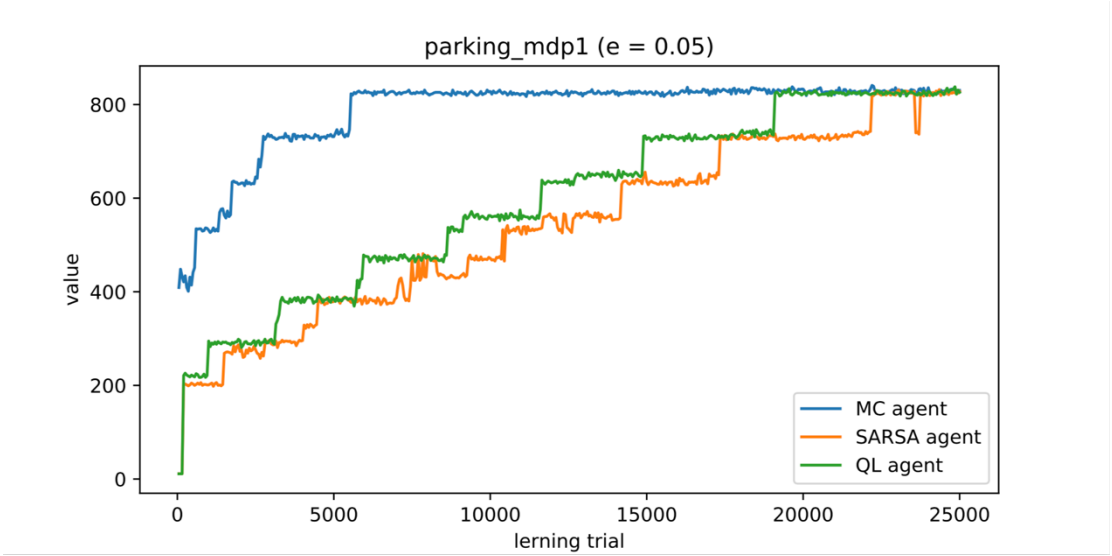
3. The result of the simple test MDP:

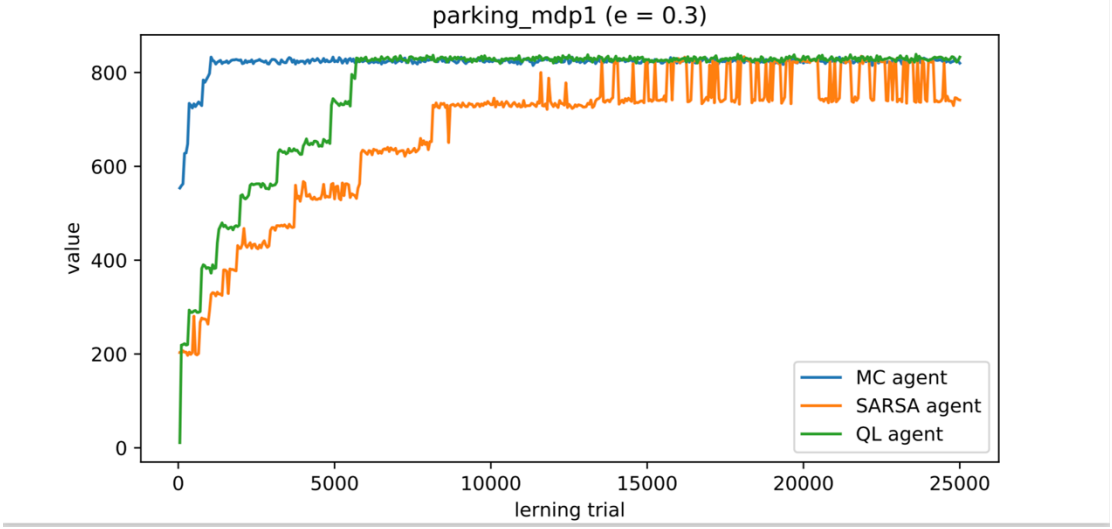




According to these result, we can see that three model-free reinforcement learning agents can find the optimal result, which is 5. The estimate of it is showed at part II.
For the MDP, the agent which has higher epsilon is faster to find the optimal results.
For three agents and epsilon equals 0.05, which means moderate exploration, Monte Carol agent is the fastest, Q-learning agent is medium, and the SARSA agent is the slowest one.
For the epsilon equals 0.3, they are closed.

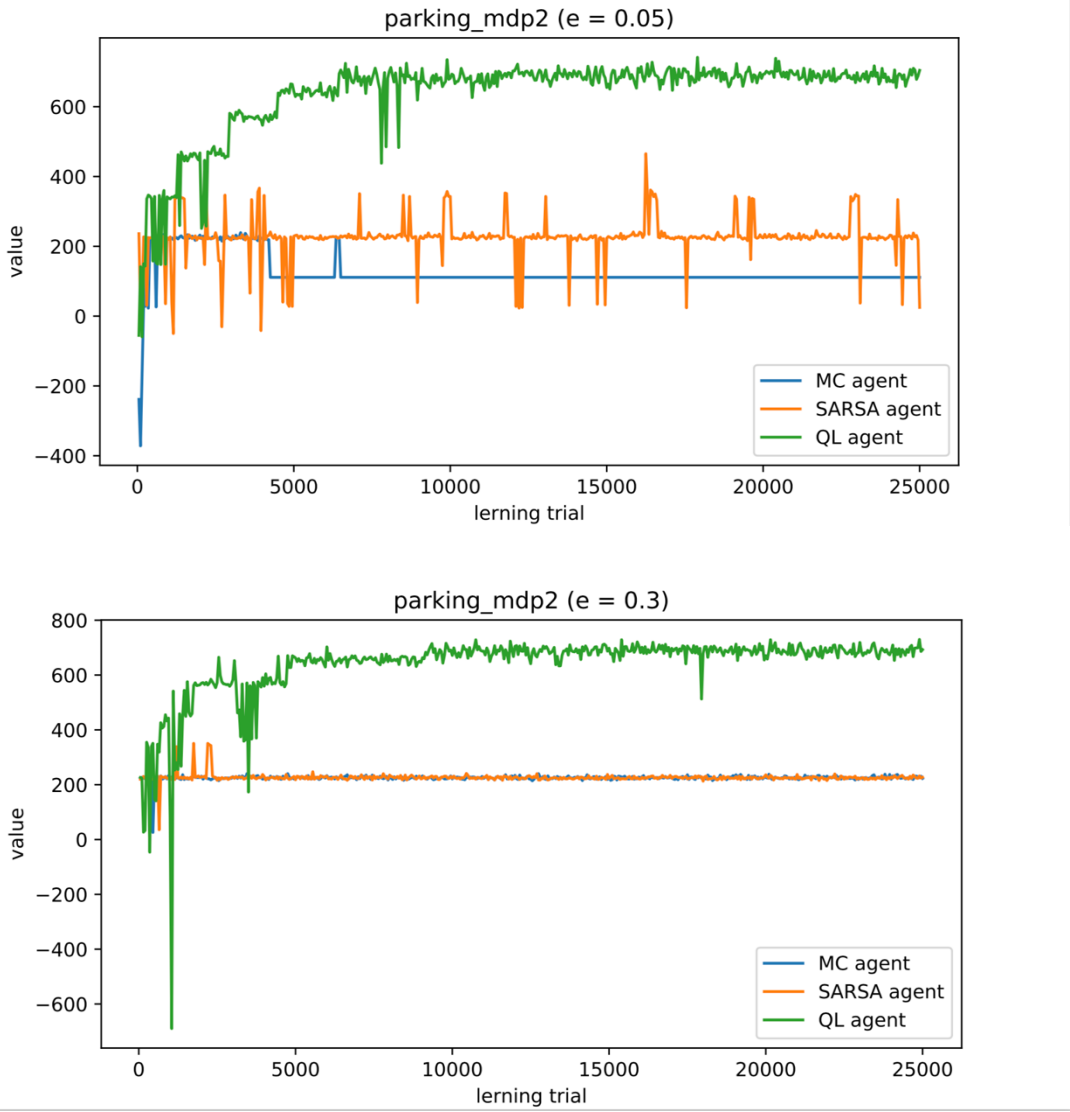
4. The result of the parking MDP1 (punishment is low):





According to the result, the agents which have higher epsilon, are faster to converge and find the optimal results.
For three agent, their ranking is same as the simple MDP. It is noteworthy that the SARSA agent is not too stable. After further learning, it still changes its actions at some state. We will see it is quite unstable at next part.

5. The result of the parking MDP2 (punishment is high):



According to the results, only the Q-learning agent can find the optimal results. For the other two, they are stop at a certain range, and be afraid of moving forward.
For the epsilon equals 0.05, the SARSA agent is quite unstable, and never reach the optimal. It always changes its action at some state, even after learning 25000 episodes. The Monte Carol can get the better reward at first 5000 learning, but the interesting thing is. After learning more episodes, it is worse, and it converge to a worse sub-optimal policy.
For the epsilon equals 0.3, Q-learning agent is still good, and faster to converge to optimal policy. The SARSA agent and Monte Carol agent is better but still cannot find the optimal results. Both of them are bad because of the negative reward. They are afraid of that.

6. To sum up, the negative reward has a little bit influence on Q-learning, it is quite stable, and can handle this situation. However, the SARSA agent and Monte Carol agent cannot handle the large negative reward MDP well. They seem are more “timid” than the Q-

learning agent, but the Monte carol agent is faster and better than Q-learning agent at the small negative or non-negative reward MDP. For the epsilon, the results showed that, higher epsilon is better than lower epsilon, but we should do more research to prove that. Maybe, we can do more research to give different learning rate, and different kinds of MDP.